Looping Commands and Working with user Inputs

CHAPTER 4

The For Command

- for command allows you to create a loop that iterates through a series of values.
- Loops are used to execute set of command for multiple times.

The For Command

- Syntax:

for var in *list*

do

commands

done

- The commands entered between the do and done statements can be one or more standard bash shell commands.

Reading values in a list

```
for i in PCP DYP GSM MMP
do
   echo The next college is $i
Done
$./test1
The next college  is PCP
The next college is DYP
The next college is GSM
The next college is MMP
```

Reading complex values in a list

```
for test in I don’t know if this’ll work
do
        echo "word:$test"
done
$ ./badtest1
word:I
word:dont know if thisll
word:work
$
```

By using \ char

```
for test in I don\’t know if "this’ll" work
do
   echo "word:$test"
done
$ ./test2
word:I
word:don’t
word:know
word:if
word:this’ll
word:work
$
```

By using “”

```
for  test in Nevada  "New Mexico" "New York"
do
echo "Now going to $test"
done
$ ./test3
Now going to Nevada
Now going to New Mexico
Now going to New York
$
```

Reading a list from a variable

```
list=“Mumbai Pune Nashik Nagpur"
list=$list" Connecticut"
for state in $list
do
echo "WE HAVE VISITED $state?"
done
$ ./test4
WE HAVE VISITED Mumbai
WE HAVE VISITED Pune
WE HAVE VISITED Nashik
WE HAVE VISITED Nagpur
 $
```

Reading values from a command

```
file="states"
for state in `cat $file`
do
        echo "Visit beautiful $state"
done
$ cat states
Maharashtra Gujarat Madhyapradesh
$ ./test5
Visit beautiful Maharashtra
Visit beautiful Gujarat
Visit beautiful Madhyapradesh
```

Changing the field separator

- The IFS (*internal field Separator)* environment variable defines a list of characters the bash shell uses as field separators.

- A space
- A tab
- A newline

Changing the field separator

```
file="states“
IFS=$’\n’
for state in `cat $file`
do
        echo "Visit beautiful $state"
done
$ cat states
Maharashtra
Gujarat
Madhyapradesh
$ ./test5 (with the same output)
```

Reading a directory using wildcards

```
For  file in /home/rich/test/*
do
        if [ -d "$file" ]
        then
                echo "$file is a directory"
        elif [ -f "$file" ]
        Then
                echo "$file is a file"
        fi
done
$ ./test6
```

The C-Style for Command

- Syntax:

for ((*variable assignment ; condition ; iteration process*))

E.g.:

for ((a = 1; a ‹ 10; a++))

To print 1 – 10 nos.

```
for (( i=1; i <= 10; i++ ))
do
    echo "The next number is $i"
done
$ ./test8
The next number is 1
The next number is 2
The next number is 3
The next number is 4
The next number is 5
The next number is 6
The next number is 7
The next number is 8
The next number is 9
The next number is 10
```

Using multiple variables

```
for (( a=1, b=10; a <= 10; a++, b-- ))
do
echo "$a - $b"
done
$ ./test9
1 - 10
2 - 9
3 - 8
4 - 7
5 - 6
6 - 5
7 - 4
8 - 3
9 - 2
10 - 1
$
```

While Command

- **Basic while format**

The format of the while command is:

while test *command*
do
other commands
done

To print 10-1 nos

```
var1=10
while [ $var1 -gt 0 ]
do
        echo $var1
        var1=$[ $var1 - 1 ]
done
$ ./test10
10
9
8
7
6
5
4
3
2
1
$
```

Using multiple test commands

```
var1=10
while echo $var1
[ $var1 -ge 0 ]
do
        echo "This is inside the loop"
        var1=$[ $var1 - 1 ]
done
$ ./test11
10
This is inside the loop
9
This is inside the loop
8
.
.
```

The until Command

- The format of the until command is:

until test *commands*
do
other commands
done

Simple program of until command

```
var1=100
until [ $var1 -eq 0 ]
do
        echo $var1
        var1=$[ $var1 - 25 ]
done
$ ./test12
100
75
50
25
```

Nesting Loops

- A loop statement can use any other type of command within the loop, including other loop commands. This is called a *nested loop*
- This is an iteration within an iteration
- which multiplies the number of times commands are being run.

Nesting Loops

```
for (( a = 1; a <= 3; a++ ))
do
        echo "Starting loop $a:"
        for (( b = 1; b <= 3; b++ ))
        do
                echo " Inside loop: $b"
        done
done
$ ./test14
Starting loop 1:
  Inside loop: 1
  Inside loop: 2
  Inside loop: 3
Starting loop 2:
  Inside loop: 1
  Inside loop: 2
  Inside loop: 3
Starting loop 3:
  Inside loop: 1
  Inside loop: 2
  Inside loop: 3
$
```

Controlling the Loop

- There are two commands that help us control what happens inside of a loop:

- They are:

1. Break Command
2. Continue Command

The break command

- Breaking out of a single loop

```
for var1 in 1 2 3 4 5 6 7 8 9 10
do
    if [ $var1 -eq 5 ]
    then
            break
    fi
    echo "Iteration number: $var1"
done
echo "The for loop is completed"
$ ./test17
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
The for loop is completed
$
```

Breaking out of an outer loop

```
for (( a = 1; a ‹ 4; a++ ))
do
echo "Outer loop: $a"
    for (( b = 1; b ‹ 100; b++ ))
    do
            if [ $b -gt 4 ]
            then
                    break 2
            fi
            echo " Inner loop: $b"
    done
done
$ ./test20
Outer loop: 1
Inner loop: 1
Inner loop: 2
Inner loop: 3
Inner loop: 4
$
```

The continue command

```
for (( var1 = 1; var1 <= 5; var1++ ))
do
        if test $var1 -eq 2
        then
                continue
        fi
echo "Iteration number: $var1"
done
$ ./test21
Iteration number: 1
Iteration number: 3
Iteration number: 4
Iteration number: 5
```

Processing the Output of a Loop

```
for file in /home/rich/*
do
        if [ -d "$file" ]
        then
                echo "$file is a directory"
        elif
                echo "$file is a file"
        fi
done > output.txt
```

For numeric data

```
for (( a = 1; a ‹ 10; a++ ))
do
echo "The number is $a"
done > test23.txt
echo "The command is finished."
$ ./test23
The command is finished.
$ cat test23.txt
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
$
```

Command Line Parameters

- Method of passing data to your shell script is *command line parameters*.
- *Command line parameters allow you to* add data values to the command line when you execute the script
- $ ./addem 10 30

Reading parameters

```
factorial=1
for (( number = 1; number ‹= $1 ; number++ ))
do
                    factorial=$[ $factorial * $number ]
done
echo The factorial of $1 is $factorial
$ ./test1 5
The factorial of 5 is 120
$
```

Reading parameters

```
total=$[ $1 + $2 ]
echo The first paramerer is $1.
echo The second parameter is $2.
echo The total value is $total.
$ ./test2 2 5
The first paramerer is 2.
The second parameter is 5.
The total is 7.
$
```

Reading the program name

```
echo The command entered is: $0
$ ./test5
The command entered is: ./test5
                        OR(by basename)
name=`basename $0`
echo The command entered is: $name
$ ./test5b
The command entered is: test5b
```

Testing parameters

```
if [ -n "$1" ]
then
        echo Hello $1, glad to meet you.
else
        echo "Sorry, you didn't identify yourself."
Fi
$ ./test7 Rich
Hello Rich, glad to meet you.
$ ./test7
Sorry, you didn't identify yourself.
$
```

Special Parameter Variables

- **Counting parameters($#):-**

echo There were $# parameters supplied.
$ ./test8
There were 0 parameters supplied.

$ ./test8 1 2 3 4 5
There were 5 parameters supplied.

$ ./test8 1 2 3 4 5 6 7 8 9 10
There were 10 parameters supplied.

$ ./test8 "Rich Blum"
There were 1 parameters supplied.
$

Grabbing all the data by $* and $@

```
echo "Using the \$* method: $*"
echo "Using the \$@ method: $@"
$ ./test11 rich barbara katie jessica
Using the $* method: rich barbara katie jessica
Using the $@ method: rich barbara katie jessica
$
```

$*

```
count=1
for param in "$*"
do
        echo "\$* Parameter #$count = $param"
        count=$[ $count + 1 ]
Done
$ ./test12 rich barbara katie jessica
$* Parameter #1 = rich barbara katie jessica
$
```

$@

- count=1
- for param in "$@"
- do
- echo "\$@ Parameter #$count = $param"
- count=$[$count + 1]
- done
- $ ./test12 rich barbara katie jessica
- $@ Parameter #1 = rich
- $@ Parameter #2 = barbara
- $@ Parameter #3 = katie
- $@ Parameter #4 = jessica
- $

Being Shifty

- The bash shell provides the **shift command** to help us manipulate command line parameters.
- The shift command shifts the command line parameters in their relative positions.
- It “downgrades” each parameter variable one position by default.

Shift Command

```
count=1
while [ -n "$1" ]
do
        echo "Parameter #$count = $1"
        count=$[ $count + 1 ]
shift
done
$ ./test13 rich barbara katie jessica
Parameter #1 = rich
Parameter #2 = barbara
Parameter #3 = katie
Parameter #4 = jessica
$
```

THANK YOU